Booleans and Conditionals

15-110 – Monday 09/09

Learning Goals

Use **programming** to specify algorithms to computers

- Use **Booleans** to compute whether an expression is True or False
- Use **conditionals** to write algorithms that make choices based on data
- Use **nesting** of statements to create complex control flow

Booleans

Booleans: values that can be True or False

In week 1, we learned about the **Boolean** type, which can be one of two values: True or False

Until now, we've made Boolean values by comparing different values, such as:

- `x < 5`
- `s == "Hello"`
- `7 >= 2`

Combining Booleans

We aren't limited to only evaluating a single Boolean expression! We can **combine** Boolean values using **logical operations.** We'll learn about three- **and, or,** and **not**.

Combining Boolean values will let us check complex requirements while running code.

And Operation

The **and** operation takes two Boolean values and evaluates to True if **both** values are True. In other words, it evaluates to False if **either** value is False.

We use **and** when we want to require that both conditions be met at the same time.

Example:

```
(x >= 0) and (x < 10)
```

a	b	a and b
True	True	True
True	False	False
False	True	False
False	False	False

Or Operation

The **or** operation takes two Boolean values and evaluates to True if **either** value is True. In other words, it only evaluates to False if **both** values are False.

a	b	a or b
True	True	True
True	False	True
False	True	True
False	False	False

We use **or** when there are multiple valid conditions to choose from.

Example:

```
(day == "Saturday") or (day == "Sunday")
```

Not Operation

Finally, the **not** operation takes a single Boolean value and switches it to the opposite value (negates it). not True becomes False, and not False becomes True.

We use **not** to switch the result of a Boolean expression. For example, not (x < 5) is the same as x >= 5.

Example:

```
not (x == 0)
```

a	not a
True	False
False	True

Activity: Guess the Result

If x = 10, what will each of the following expressions evaluate to?

```
print(x < 25 and x > 15)

print(x < 25 or x > 15)

print(not (x > 5 and x < 10))

print((x > 5) or ((x**2 > 50) and (x == 20))

print(((x > 5) or (x**2 > 50)) and (x == 20))
```

Conditionals

Conditionals make Decisions

With Booleans, we can make a new type of code called a **conditional**. Conditionals are a form of a **control structure**- they let us change the direction of the code based on the value that we provide.

To write a conditional (or **if statement**), we use the following structure:

```
if <Boolean value>:
    <conditional body>
```

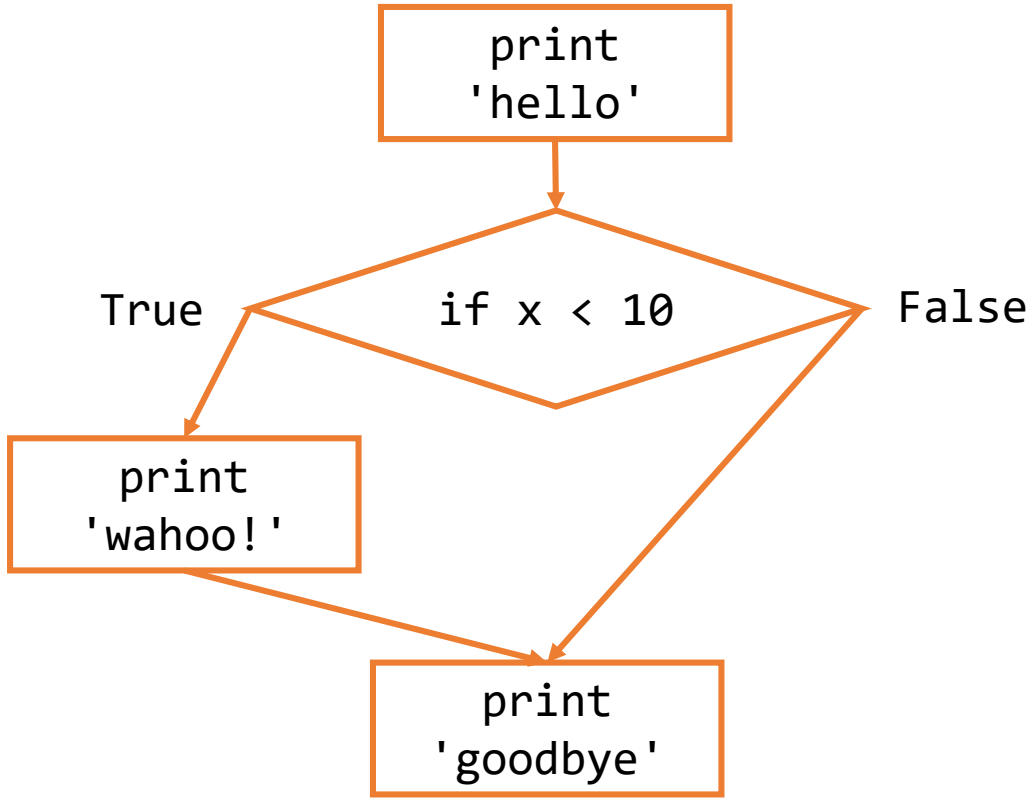
Note that, like a function definition, the top line of the if statement ends with a colon, and the body of the if statement is indented.

Flow Charts Show Code Choices

We'll use a **flow chart** to demonstrate what happens to the code of an if statement based on the values provided.

```
print("hello")
if x < 10:
    print("wahoo!")
print("goodbye")

# wahoo! is only printed
# if x is less than 10.
# But hello and goodbye
# are always printed.
```



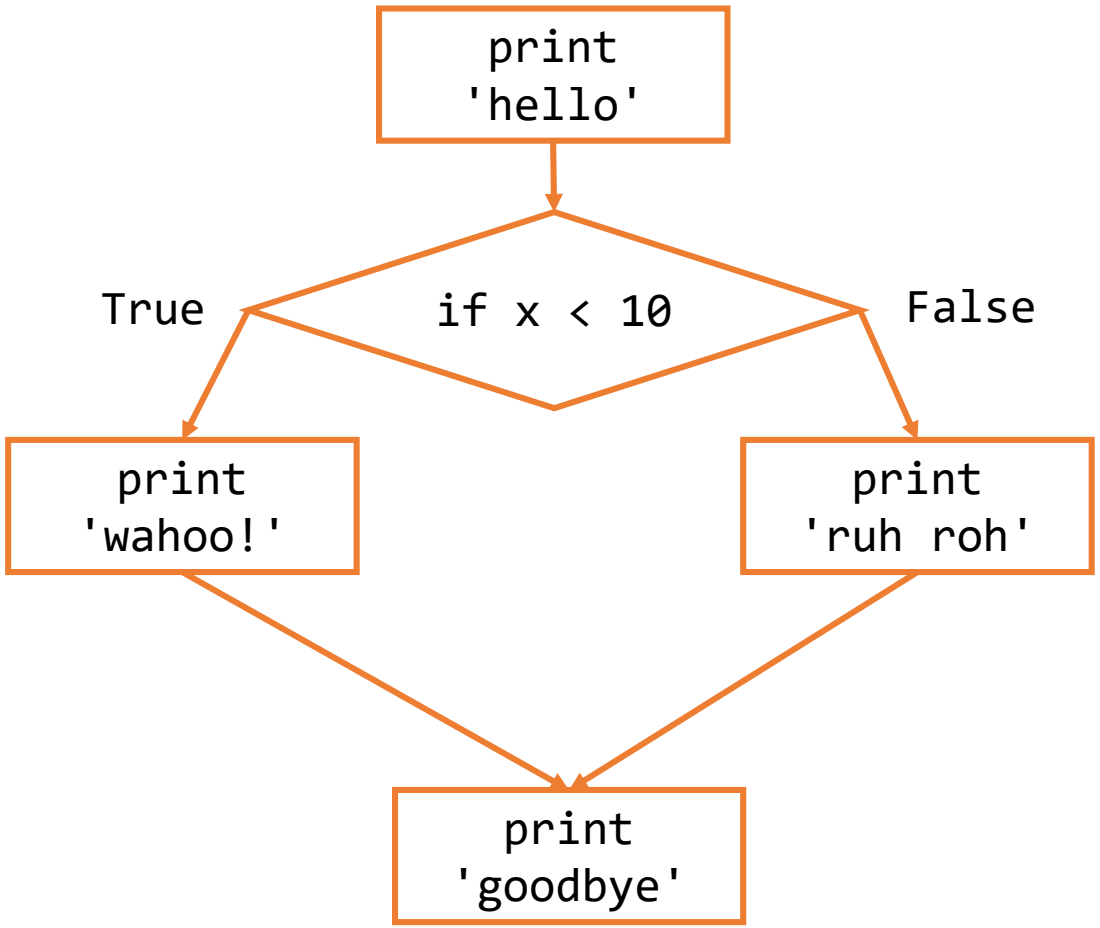
Else Statements allow Alternatives

Sometimes we want a program to do one of two possible actions based on the conditions. In this case, instead of writing two if statements, we can write a single if statement and give it an **else**. The else will cover the case when the Boolean expression is False.

```
if <Boolean_expression>:
    <body_if_true>
else:
    <body_if_false>
```

Updated Example

```
print("hello")
if x < 10:
    print("wahoo!")
else:
    print("ruh roh")
print("goodbye")
```



Conditional Example

Prediction Exercise: What will the following code print?

```
x = 5
if x > 10:
    print("Up high!")
else:
    print("Down low!")
```

Question: What could we change to get the other statement to print instead?

Question: Can we get the program to print out both statements?

Else Must be Paired with If

It's impossible to have an else statement by itself, as it would have no condition to be the alternative to.

Therefore, **every else must be paired with an if**. Likewise, every if can have **one else at most**.

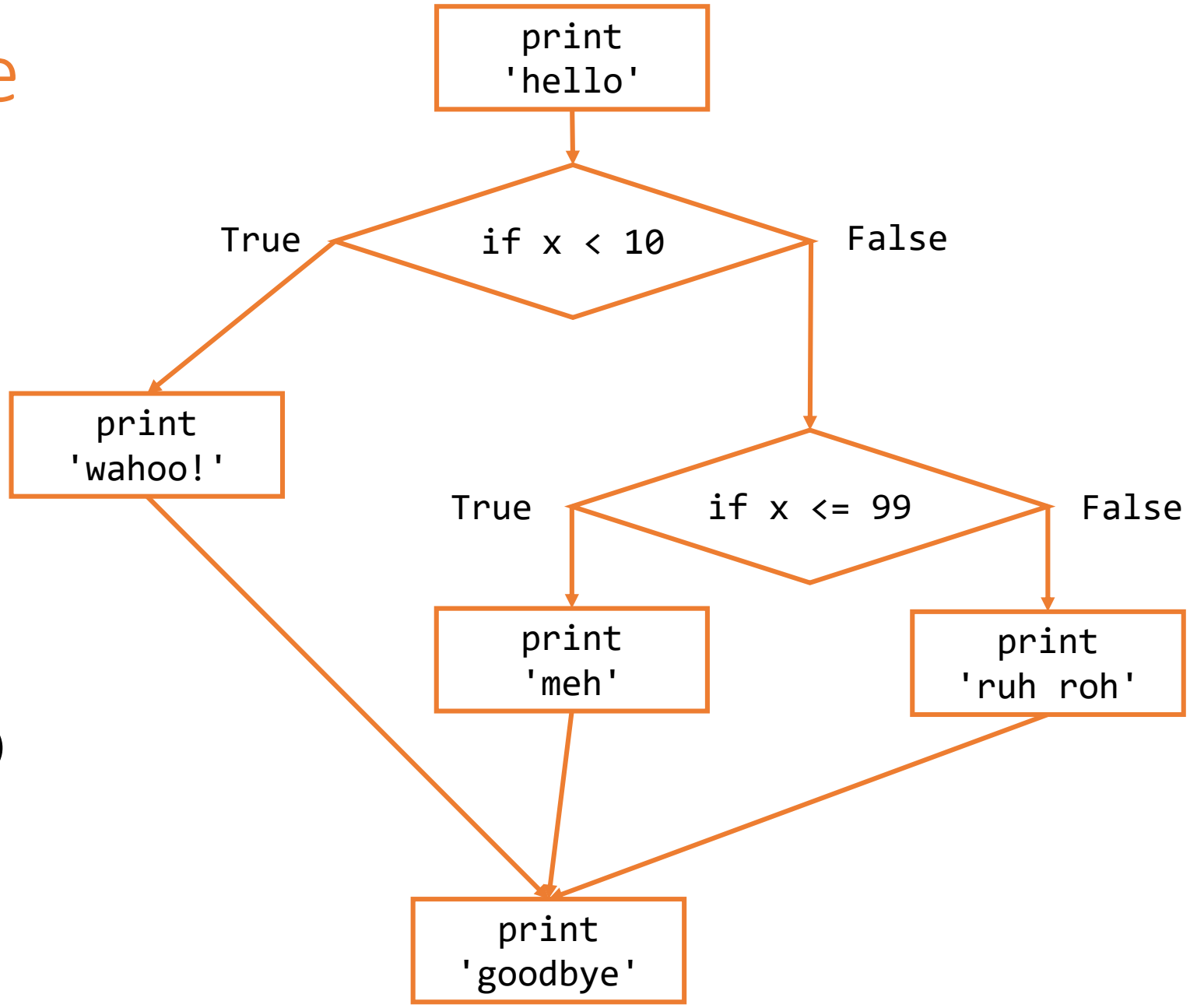
Elif Implements Multiple Branches

Finally, we can use **elif** statements to add alternatives to an if statement that have their own conditions. An elif is like an if statement, except that it is only checked **if the if statement evaluates to False**.

```
if <Boolean_expression_A>:
    <body_if_A_True>
elif <Boolean_expression_B>:
    <body_if_A_False_and_B_True>
else:
    <body_if_both_False>
```

Updated Example

```
print("hello")
if x < 10:
    print("wahoo!")
elif x <= 99:
    print("meh")
else:
    print("ruh roh")
print("goodbye")
```



Conditional Blocks

We can have more than one elif branch associated with an if statement. In fact, we can have as many as we need! But, like with else statements, an elif must be associated with an if statement (or a previous elif).

In general, a **conditional block** is an if statement, 0+ elif statements, and 0-1 else statements that are all joined together. These can be considered a single control structure.

Example: gradeCalculator

Let's write a few lines of code that asks the user to input a grade as a number, then calculates the letter grade that corresponds to that number.

90+ is an A, 80-90 is a B, 70-80 is a C, 60-70 is a D, and below 60 is an R.

Nesting Control Structures

Nesting Creates More Complex Control Flow

Now that we have a control structure, we can do more than just use single lines of code in the if/elif/else statement's body. In fact, **we can put if statements inside of if statements**.

In general, we'll be able to **nest** control structures inside of other control structures. We can also nest control structures inside of function definitions.

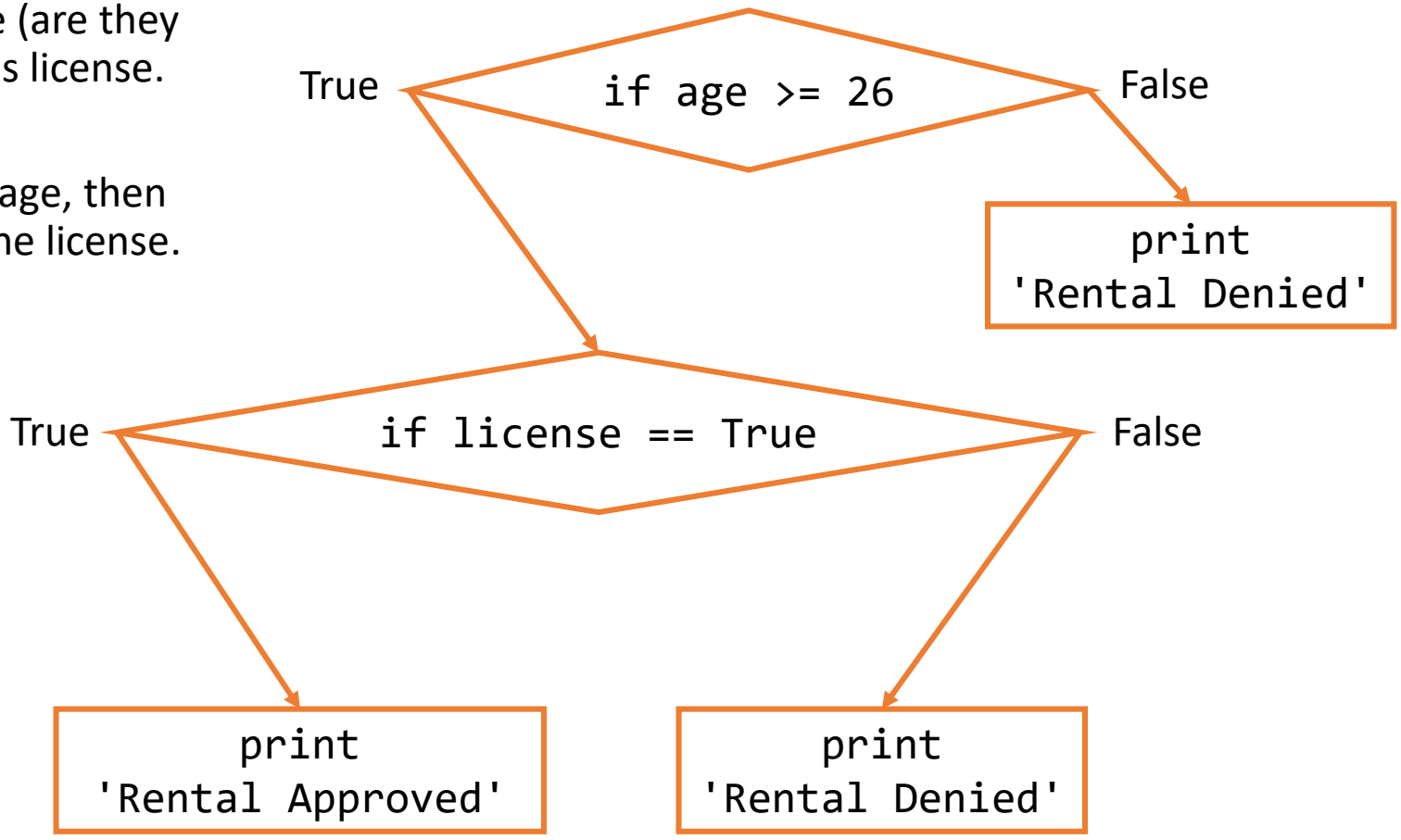
In program syntax, we demonstrate that a control structure is nested by **indenting the code** so that it's in the outer control structure's body.

Example: car rental program

Let's say we want to write a program that determines if a person can rent a car based on their age (are they at least 26) and whether they have a driver's license.

We can use one if statement to check their age, then a second (nested inside the first) to check the license. We'll only print 'Rental Approved' if both if statements evaluate to True.

```
if age >= 26:
    if license == True:
        print("Rental Approved")
    else:
        print("Rental Denied")
else:
    print("Rental Denied")
```

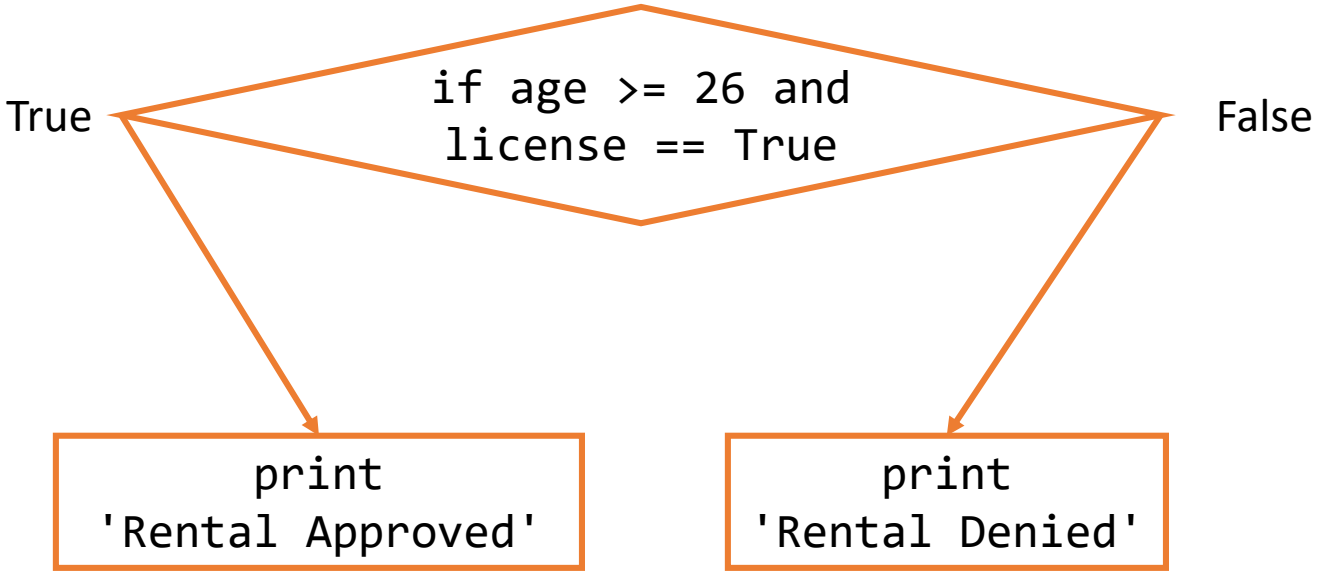


Alternative Car Rental Program

In this program, we could accomplish the same result with the and operation.

This won't always work, though- it depends on what you want the output to look like.

```
if age >= 26 and license == True:
    print("Rental Approved")
else:
    print("Rental Denied")
```



Nesting and Elif/Else Statements

When we're pairing up elif or else statements in nested code, we'll pair them with the if statement at the **same indentation level**. This is true even if an inner if statement comes in between them! However, an outer if/elif/else statement **cannot** come between them.

```
if first == True:
    if second == True:
        print("both true!")
else:
    print("first not true")
```

Question: if we want to add an else statement to the inner if, where should it go?

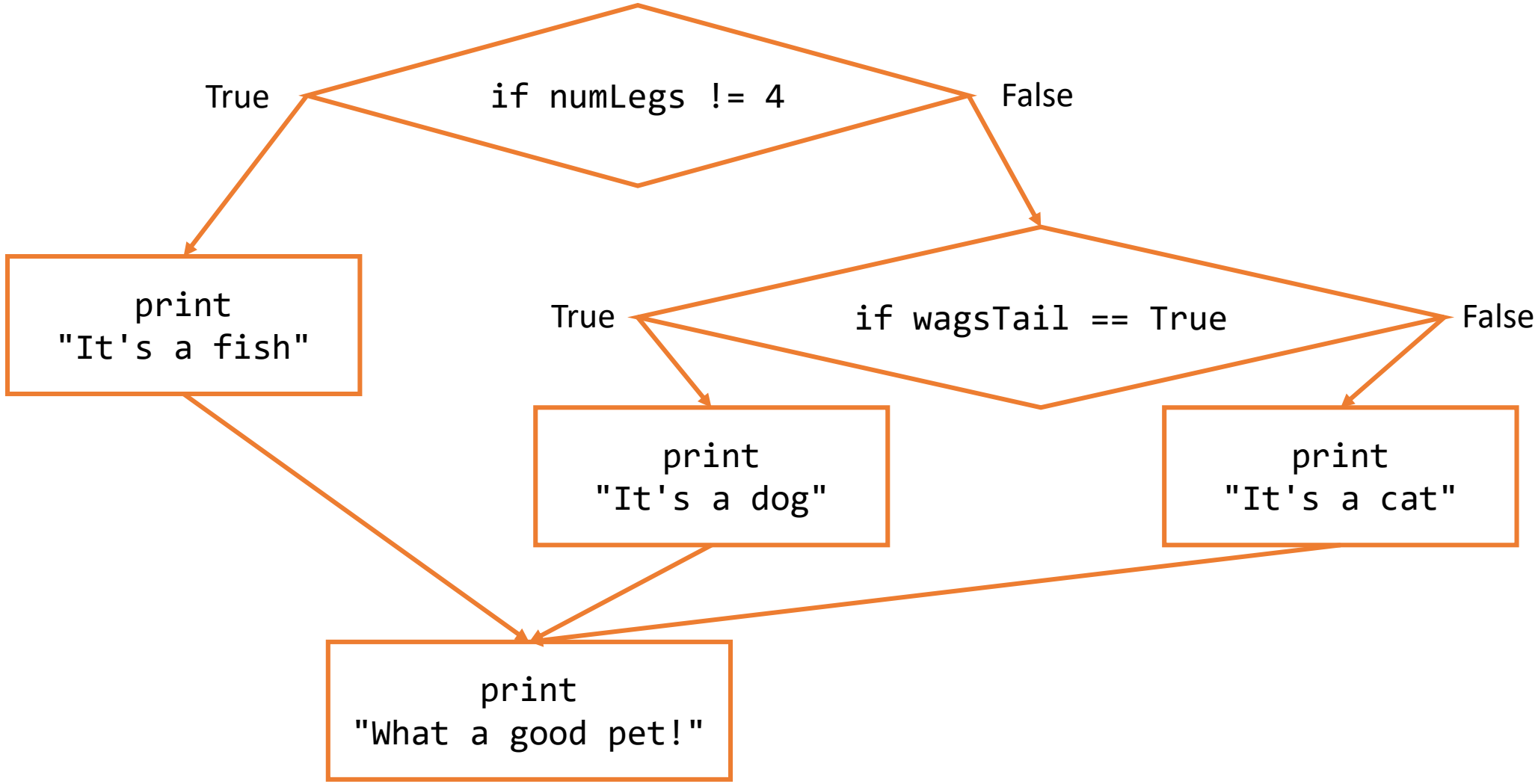
Nesting Conditionals in Functions

When we nest a conditional inside a function definition, we can **return values early,** instead of only returning on the last line. This is fine as long as we make sure every possible path the function can take will eventually return a value!

A function will always end as soon as it reaches a return statement, even if more lines of code follow it. For example, the following function will not crash when run on a negative number.

```
def squareRoot(x):
    if x < 0:
        return "Imaginary Result"
    return x ** 0.5
```

Exercise: Convert Flow Chart to Code



Learning Goals

Use **programming** to specify algorithms to computers

- Use **Booleans** to compute whether an expression is True or False
- Use **conditionals** to write algorithms that make choices based on data
- Use **nesting** of statements to create complex control flow